3. How IDOR works?

IDOR (Insecure Direct Object Reference) works by exploiting the application's failure to verify whether a user has permission to access or manipulate an internal object (like files, user data, or other resources) based on a given identifier. This oversight allows attackers to modify the identifier to access resources they shouldn't be able to reach. Here's a step-by-step breakdown:

1. Direct Access to Internal Objects

- In an IDOR vulnerability, the application uses direct identifiers (like `user_id`, `file_id`, `order_id`) in URLs or request parameters.
- When a user wants to access a resource, they might be given a URL that looks like this:

```
https://example.com/account?user_id=123
```

- This ID (`user_id=123`) directly references the object in the backend database.

2. User Manipulation of Identifiers

- If the application doesn't check whether the `user_id` belongs to the logged-in user, an attacker could change the ID to another value:

```
https://example.com/account?user_id=456
```

- This would attempt to access account data for `user_id=456`, potentially exposing another user's information.

3. Exploring Resources Without Proper Authorization

- Attackers often test other IDs by incrementing, decrementing, or using numbers they believe may exist based on their understanding of the application's structure.
- If the application doesn't enforce access controls, the attacker might view or modify sensitive data by guessing other IDs, gaining unauthorized access.

4. Lack of Access Control Verification

- An IDOR occurs because the application lacks sufficient **access control checks** on the backend. It assumes that the user is only accessing objects they're authorized to see, based purely on the ID in the request.
- Proper implementation would require the application to check if `user_id=123` belongs to the logged-in user and deny access if it does not.

Example Scenario: Accessing Another User's Data

Let's imagine a shopping application where users can view their orders:

1. **Normal Access**:

- A logged-in user views their order details with:

  ```
  https://shop.com/order?order_id=1001
  ```

- If `order_id=1001` is validated as belonging to the user, it works as expected.

2. **Exploit Attempt**:
   - An attacker guesses another order number and changes the URL:

     ```
     https://shop.com/order?order_id=1002
     ```

   - If the application does not verify ownership of `order_id=1002`, it displays another user's order details.

Preventing IDOR Vulnerabilities

To secure applications against IDOR:

- **Always Check Ownership**: Ensure every request checks that the resource belongs to the logged-in user.
- **Limit Direct Access**: Use indirect or random identifiers (e.g., UUIDs) instead of sequential numbers that are easy to guess.
- **Enforce Access Controls**: Implement server-side access controls to validate permissions before granting access to any resource.

In summary, IDOR flaws occur when applications assume users are authorized to access a resource solely based on an identifier, leading to unauthorized access if that identifier is altered.